Facebook Comment Volume Report

This report is showing the implementation of linear regression on a given dataset by applying gradient descent to minimize the error and predict the response variable as accurate as possible. We are going to use Facebook Comment Volume dataset available for download here.

As the name suggest, this dataset contains information related to user comments on the Facebook and the job is to design a regression model to model the number of comments a post would receive in H hours. This is a Multiple Linear Regression problem since it has multiple features (variables).

Model Representation

Like Simple Linear Regression, we have input variable(X) and output variable(Y). But the input variable has n features. Therefore, we can represent this linear model in general form as follows;

$$Y = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Where x_n is the i^{th} feature in input variable. By introducing $x_0 = 1$, we can write this equation.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

And converting it to matrix form has its own benefits in Python.

$$Y = \beta^T X$$

We must define the cost of the model. The cost function basically gives the error in our model. Y in above equation is our hypothesis(approximation) and we are going to define it as our hypothesis function as follows;

$$h_\beta(x) = \beta^T X$$

And the cost is,

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})^2$$

By minimizing this cost function, we can find β. We use Gradient Descent for this.

Gradient Descent

Gradiesnt Descent is an optimization algorithm. We will optimize our cost function using Gradient Descent Algorithm becomes small enough (i.e. convergence takes place).

Step 1

Initialize values $\beta_0, \beta_1, \dots, \beta_n$ with some value. **I have initialized them with 0 initially**.

Step 2

Iteratively update until it converges,

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

Here α is the learning rate and $\frac{\partial}{\partial \beta_j} J(\beta)$ means we are finding partial derivate of cost w.r.t each β_j. In step 2 we are changing the values of β_j in a direction in which it reduces our cost function. And Gradient gives the direction in which we want to move. Finally, we will reach the minima of our cost function. But we don't want to change values of β_j drastically, because we might miss the minima. That's why we need learning rate.

Implementation

I have used ***Python*** as a language and ***Jupyter notebook*** to write the code. Though it looks a lengthy and time-consuming task, I have made use of vectors which makes the process easy. First let's take an overall look at our dataset by performing exploratory analysis.

Exploratory Analysis – EDA

The main objective of performing EDA is to get a gist of your data before-hand. For this I have plotted heatmaps and pair plots to understand the overall relationship between variables/features with the Target variable. Above all, I have done the necessary imports of the python packages that will help us perform various EDA techniques.

The below heat map is shown because it had most of the variation in the data. As you can see, CC1, CC2,

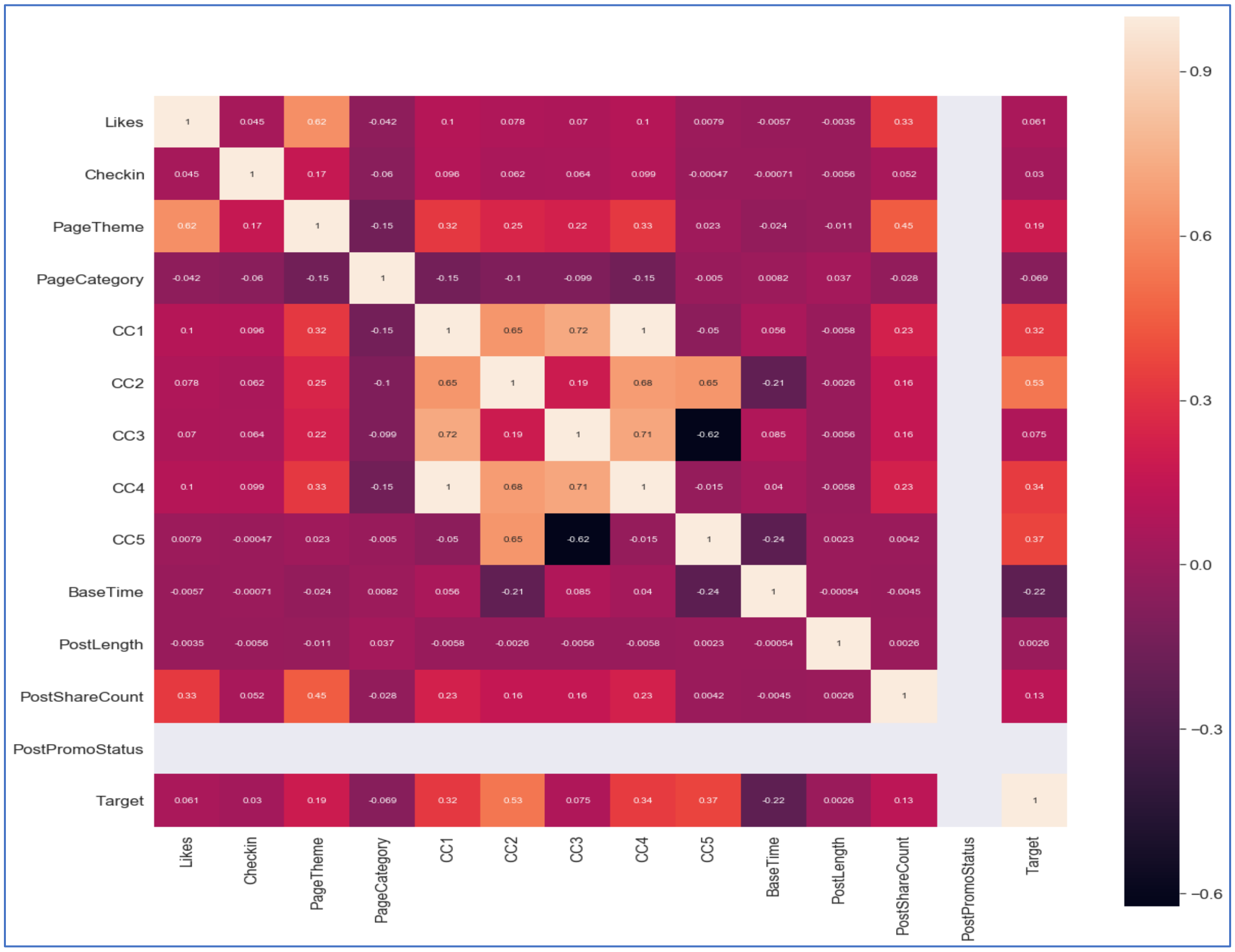

Fig 1. Heat map

and CC4 shows linear relation with Target variable as compared to others. This means they stand a good chance of depicting relation with the Target variable. From the Heat map it's evident that the correlation between response and independent variables is not so great. But this behavior could be because of the choice of the variables and due to less data points available. Moving on; let's begin with the experiments.

In all the experiments that I have performed, I have used '*Training variant 4*' as my dataset to train the model. The steps followed are as follows:

1. I began with importing it into the workspace using the pandas read_csv function.
2. Dropped some unnecessary variables based on the decision by viewing the correlation matrix
3. Splitted the dataset into 80:20 train, test respectively
4. Separated the independent and response variable into separate arrays
5. Scaled the 80% independent variables (without the target variable)
6. Generated beta with initial values 0 as array
7. Implemented cost and gradient descent functions
8. Iterated the beta values for number of iterations to achieve convergence (by reducing the cost function)
9. Plotted the cost function of train data across number of iterations as well as cost function of test data across the same iterations

Throughout the experiments the process of splitting, normalizing and training the data sets remains the same.

Experiment 1: Experiment with various values of learning rate ∝ and report on your findings as how the error varies for train and test sets with varying ∝. Plot the results. Report your best ∝ and why you picked it.

After training the data for different values of ∝, we get the following results for train and test datasets. The figure below clearly shows us, on increasing the alpha (learning rate) value for both training and test datasets the error function (i.e. cost function) decreases. It decreases gradually for smaller values of alpha as seen in the first four figures (from the top) and for the others it decreases very quickly and sharply. This behavior is observed due to higher values of alpha as it forces the model to make bigger steps to reach the global minima.

The cost function value of alpha = 0.01 and 0.1 are approximately similar. Hence, taking alpha value of 0.01 looks more promising because it is taking adequate steps to reach the global minimum assuring more accurate value when number of iterations will be increased. With alpha value higher, there is a possibility that the cost function may shoot the global minimum value and keep iterating in the loop. Therefore, I have chosen the **alpha = 0.01.**

Experiment 2: Experiment with various thresholds for convergence. Plot error results for train and test sets as a function of threshold and describe how varying the threshold affects error. Pick your best threshold and plot train and test error (in one figure) as a function of number of gradient descent iterations

The threshold value is an important parameter because it defines when to stop computing the cost function viz. it denotes that the convergence has been achieved while training the data. Fig 3. Shows the varying threshold and respective change in the error function values both together and separate. It is clear that

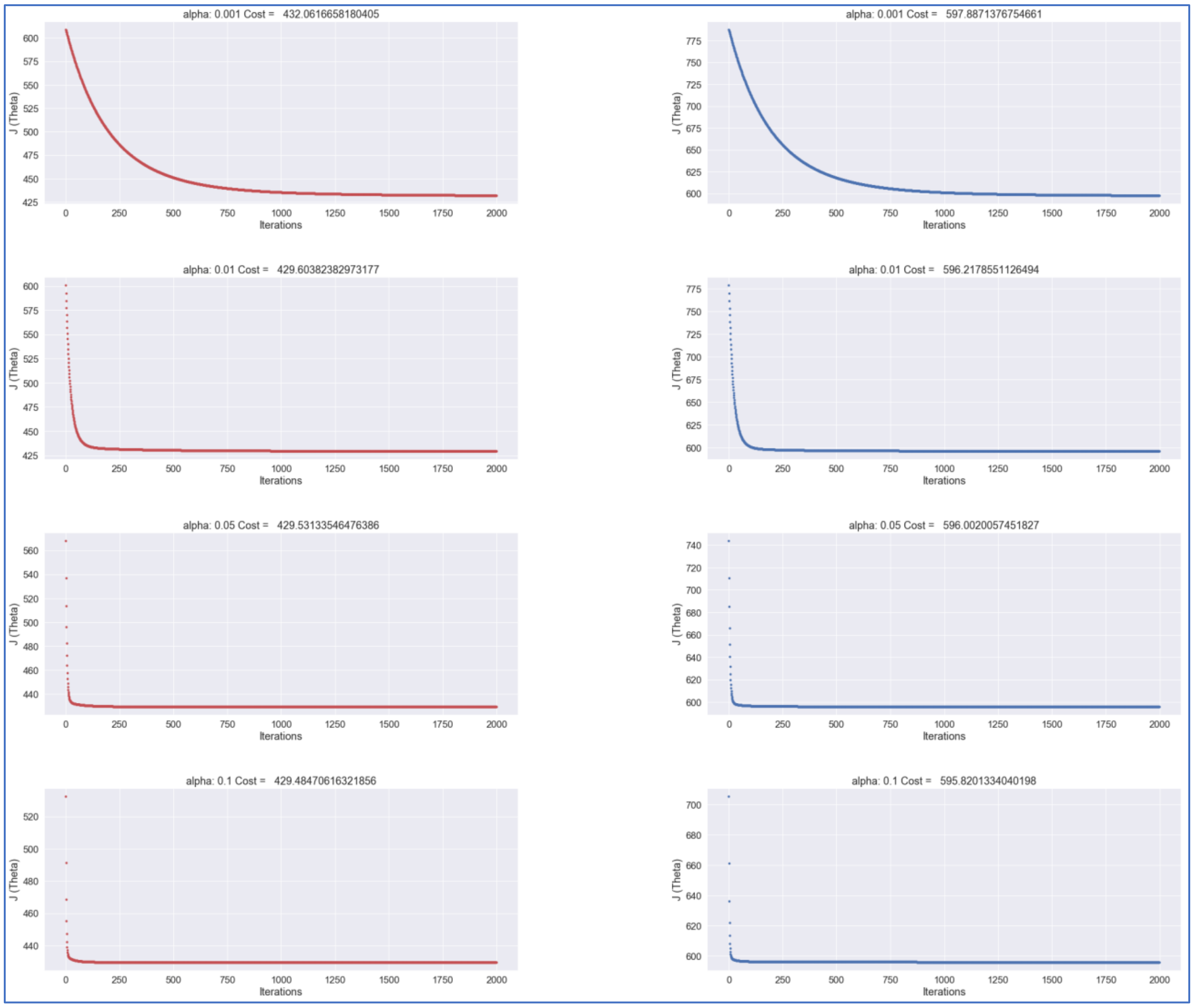

Fig 2. Error plot for train and test data

the error decreases here as well. But one thing to note is that the threshold value acts like a zooming factor to your graph. It doesn’t let the unnecessary iterations takes place if the convergence has been achieved or likely to achieve earlier in the iteration.

The third column figures are the representation of the decreasing error values as the threshold changes. Clearly, it is visible that the error value of training data set for all the three cases is lesser than the testing data set. The reason being that we have trained the model on the training data and thus it performs better on this data set whereas the testing data set has unknown values and hence the error is more, but it also decreases gradually.

We also see that keeping the threshold values higher results into forcing the convergence to happen at an earlier stage which could be a drawback and would like to avoid it. Ideally, the error value should decrease and be as close as possible to the actual response variable. The best threshold value seems to be 0.1 because it gives a lesser cost value viz **410.25.** The different threshold values used were (1,0.1,0.01).

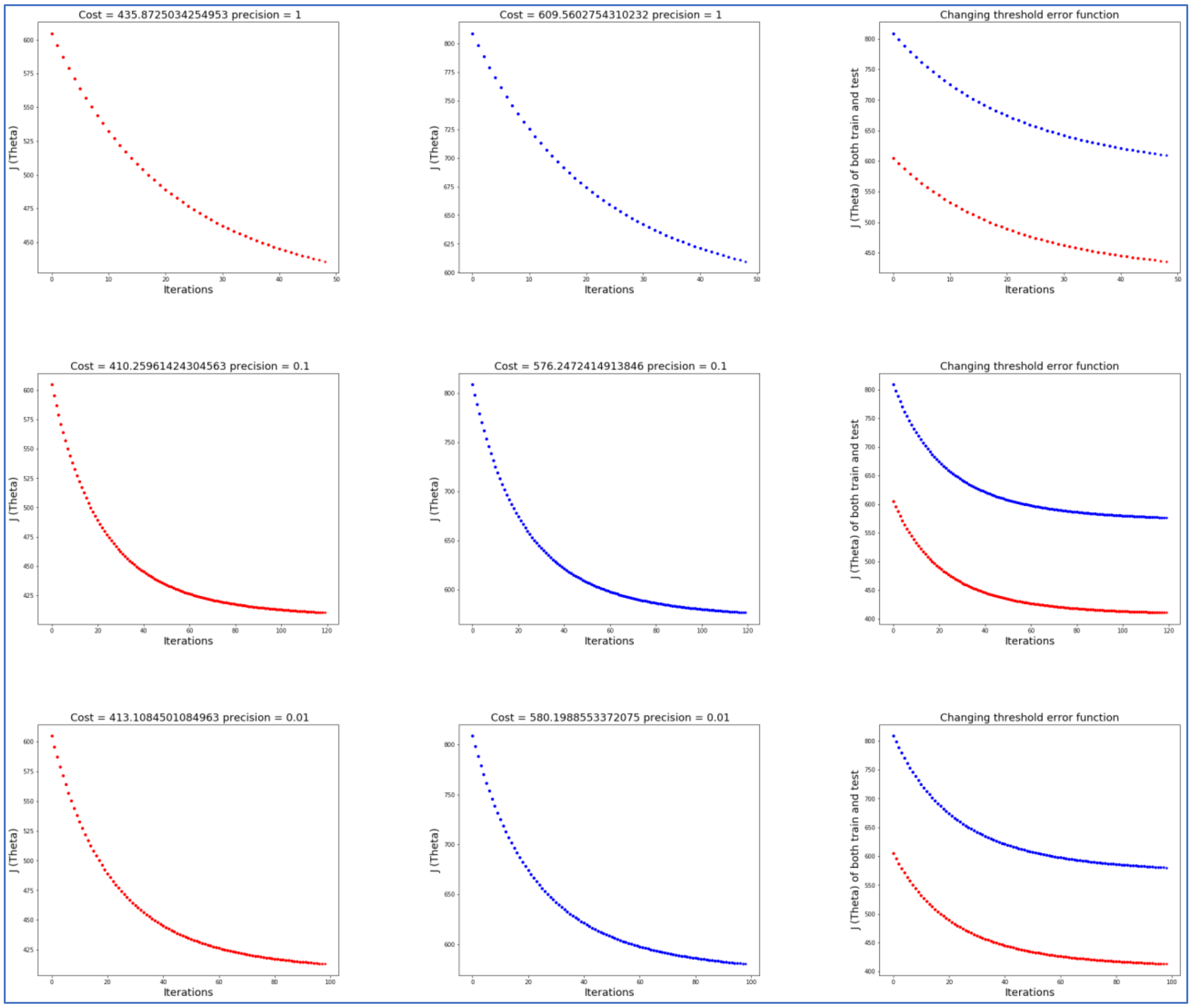

Fig 3. Error function for changing thresholds

Experiment 3: Pick five features randomly and retrain your model only on these five features. Compare train and test error results for the case of using your original set of features (greater than 10) and five random features. Report which five features did you select randomly

Model Equation for this experiment is:

$$Y = \beta_0 x_0 + \beta_1 * Likes + \beta_2 * Checkin + \beta_3 * PostPromoStatus + \beta_4 HLocal + \beta_5 PostPublishSun$$

I have selected the 5 random variables called 'Likes','Checkin','PostPromoStatus','HLocal','PostPublishSun' along with the Target variable and trained the model with the steps mentioned previously. The minimum cost value of the training data comes out to be 410.25 whereas the minimum cost value of the original data set is 429.48. This means the dataset with randomly selected variables has lower value of cost function.

This is no coincidence because if you noticed, the selected random variables in the dataset doesn't have much of the data points recorded. Due to this, there is no statistical significance that lesser value of cost function corresponds to a good model. Moreover, the heat map at the start shows that the correlation of these variables with the target variable is poor which explains the same. The hyper parameters are alpha = 0.01 and threshold 0.1. The updated beta array is as shown below:

newBeta = [5.00154257, -0.16460971, -0.07562439, 1.90969345, -0.45978836, 2.895038, 7.4115316 , 0.09796725, 3.10389123, 5.80994994, -3.95379042, 0.1060982 , 0.82078834, 0.46783174]

Experiment 4: Now pick five features that you think are best suited to predict the output, and retrain your model using these five features. Compare to the case of using your original set of features and to random features case. Did your choice of features provide better results than picking random features? Why? Did your choice of features provide better results than using all features? Why?

Having worked on the previous experiments in this report and with the help of exploratory analysis, the perfect five variables according to me could be 'Likes','Checkin','CC1','CC4','HLocal' along with the Target variable. The hyper parameters are alpha = 0.01 and threshold = 0.1 since they were selected from the previous two experiments.

After training the model with these parameters, I am getting a cost value of 410.25 which is lesser in magnitude than the one with my first experiment viz. 429.48s. The initial beta values passed were [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.] and the final values obtained are [5.00154257, -0.16460971, -0.07562439, 1.90969345, -0.45978836, 2.895038, 7.4115316, 0.09796725, 3.10389123, 5.80994994, -3.95379042, 0.1060982, 0.82078834, 0.46783174].

To compare overall, $perfect5(cost\ error) < random5(cost\ error) < morethan10(cost\ error)$. The choice of my variable selection turned out to be fruitful in achieving a lower cost error. The reason being that these 5 variables have a good correlation factor with the target variable and these variables are depicting behavior about the target variable.

Not all the other variables don’t have a good relationship with the response variable, and thus they are dropped at first instance from the modelling. Some even had negative relationship and it was better to exclude them from the future operations.

Describe your interpretation of the results. What do you think matters the most for predicting the number of comments? What other steps you could have taken with regards to modeling to get better results?

I feel the content of the post has more weightage in drawing attention from the users online. If the users feel themselves engaged in the post, the percentage of people commenting could increase. The results obtained during these experiments were quantifiable and were a challenge to interpret since I was new to this. I feel it could have been a logistic regression problem as well in predicting the days were you get more comments.